Introduction

The percentage of human activity that takes place in the physical world of earth, wind, fire, and water has been steadily decreasing over the past half century. This decrease has been accompanied by a concomitant increase in the percentage of activity that takes place in cyberspace.

This wave of ever-increasing computerization has been lapping at the shore of medicine for over two decades and now promises to envelop it in chest-deep water. This chapter briefly describes an application of computer science methods to developing a cutting-edge system that helps surgeons plan and execute neurosurgery. A particular component of this method, an artificial intelligence algorithm which aligns two different views of the patient's head, is discussed in detail. This description serves as a springboard for a wide-ranging discussion of how computerization may effect medicine: Will medicine be buoyed by the computerization wave or drowned?

But first let us consider how medicine currently approaches disease. When a patient enters a hospital, a medical practitioner first diagnoses the patient. If the patient is ill, then a treatment plan is selected and executed. The effect of this plan upon the patient is monitored and, if need be, the plan is modified. If the plan succeeds, the patient is usually released. If the patient dies, the plan has failed, and a post-mortem analysis is performed. A flow chart diagram of this process is shown in Figure 1.

Computer aided medicine programs can support or automate any stage of this procedure. A wide variety of systems, all of them limited to a small number of pathologies, perform diagnosis. One of the best known diagnosis programs is MYCIN which specializes in certain types of infectious diseases. MYCIN is notable for its use of if-then rules and its domain-independent architecture that has supported the implementation of other programs, such as PUFF [1], that can diagnose different diseases. Indeed, the domain-independent component of MYCIN, called EMYCIN ("Essential MYCIN"), has been used to implement systems that have expertise in areas unrelated to medicine. This domain-independent component was the first expert system shell. Two other programs, TEIRESIAS, a knowledge acquisition system, and Guidon, a medical tutor, employ the same underlying rule-based data structure as MYCIN and expand its capabilities.

Many diagnosis programs also recommend treatments for the conditions that they identify in patients. The CASNET system [11], which has a glaucoma knowledge base, associates diagnoses with treatment plans. One important contribution of CASNET is the effective incorporation of a causal model that connects pathophysiological states.

Several systems, often embedded in medical instrumentation, monitor the conditions of a patient during treatment. Although these systems rarely employ the sophisticated data structures or algorithms of programs like MYCIN and CASNET, they reduce demands on care givers by alerting them to patients who are in critical or unexpected states.

In addition, there are computer programs that help in the operation of hospitals, in the management of patient billing records, and in various other support capacities that are not directly linked with the care of diseased individuals.

The guided imagery system described next helps in the selection and implementation of a treatment plan.

Image Guided Surgery

The image guided surgery system, which is still under development and is expected to aid neurosurgeons, can be divided into six components:

- Medical sensors, created by the General Electric corporation, help to create three dimensional models of the patient.
- Segmentation algorithms, developed by Brigham and Women's hospital, partition the images into biologically meaningful components.
- Laser scanners, the product of a collaboration between the Technical Arts Corporation and the Analytic Sciences Corporation, produce a set of points that are on the surface of the patient's skull.
- Recognition algorithms, developed at the MIT Artificial Intelligence Laboratory, align the models of the patient made by the medical sensors and the laser scanners.
- Path planning methods use traditional artificial intelligence algorithms and the data about the patient's anatomy (collected by the medical sensors) to plan the trajectory of surgical tools in order to maximize the probability of success of the operation.
- Computer visualization methods combine the data from various medical sensors and reduce the amount of invasive surgery, thereby decreasing patient discomfort.

One of the crucial components in this system is the alignment of laser scanning data with an MRI (magnetic resonance image) or CT (computer tomography) scan. The MRI or CT scan is taken before the operation and helps the surgical team plan the operation. Immediately prior to surgery, a laser scan is taken of the patient's head while the patient is on the operating table and this view of the patient needs to be aligned with the original MRI or CT scan. This alignment requires a three-dimensional rotation and a three-dimensional translation, and is performed by an artificial intelligence algorithm called dynamic hill climbing [14], which is explained in the next section.

Dynamic Hill Climbing

The dynamic hill climbing optimization algorithm is a general method for locating the minimum of a mathematical function. The problem of minimizing a function can be stated rigorously:

> Given a function, f, find the point $\vec{x}$ such that for every $\vec{y}$, $f(\vec{x}) \leq f(\vec{y})$.

Note that a maximization problem can be turned into a minimization problem by multiplying the function, f, by -1.

Although there are no methods that guarantee the location of the minimum, many techniques, including genetic algorithms [9, 7] and simulated annealing [10, 4], have been developed that in practice have excellent performance on a wide variety of problems. These algorithms do not always find the optimum (minimum), but they often find points that are close to it.

The dynamic hill climbing algorithm, when implemented in computer software, consists of an inner loop and an outer loop. The inner loop is a very efficient local optimization algorithm, while the outer loop restarts the inner loop in different parts of the search space in order to improve the chances that the global optimum will be found.

The inner loop maintains a set of $2n$ vectors (n is the dimensionality of the function, f) called orthogonal vectors. So, for example, in a two dimensional Cartesian space the directions of these four vectors would be: (1,0), (0,1), (0, -1), and (-1,0). In addition, two other vectors, called gradient vectors, are updated dynamically by the algorithm. These two vectors always point in opposite directions, although they may have different magnitudes, and their role in the algorithm is to point in the direction of steepest descent. The

Lisp code in the appendix gives an implementation of the inner loop.

The outer loop keeps track of the points visited by the inner loop, and when the inner loop halts because it has found a local optimum, the outer loop re-initializes it with a point that is far away from the ones that it has already visited. In this way, coverage of the entire space is improved.

For this application to medicine, the dynamic hill climbing algorithm minimizes the mean squared error between the laser scanning data and the MRI or CT scan data, a task which involves finding a three-dimensional rotation and a three-dimensional translation. Figure 2 shows an alignment of laser scan data (blue dots) with MRI data (orange skull). Dynamic hill climbing has been found to outperform genetic algorithms and simulated annealing on a set of test functions [5].

The Future

The previous section has sketched a cutting-edge application of computer science methods to medicine. However, this system, even in its most imaginative and far-flung implementation, only incrementally changes the surgeon's task. This is true of all current computer aided medicine programs and has been true for over twenty years. Indeed, in 1970, Schwartz [12], one of the most astute observers in the field, wrote:

> The computer thus remains...an adjunct to the present system [of medical care], serving a palliative function, but not really solving the major problems of the system.

Unfortunately, the same statement is true today. In this section we consider potentially radical changes in the practice of medicine that we expect will be precipitated by new computer science methods.

Changing incentives

Many have noted that medical practitioners hesitate to embrace computerization. They have attributed this fact to poor user interfaces [13], inadequate communication pathways [2], and inappropriate incentive structures [8] (explained in the next paragraph). Some have also suggested that there is an entrenched bias against any use of computers, even if they demonstrably improve patient care.

We feel that inappropriate incentive structures are the most compelling and the most troublesome of these problems. The equation is fairly simple:

more prescriptions, more surgery, and more tests mean more money. Indeed, if a patient is cured, a source of income is eliminated. Hence, the main interest of doctors and hospitals is directly opposed to the interest of the patient.

Grossman [8] notes this and suggests that the current wave of cost control in hospitals will reduce this tendency to prescribe ever more expensive drugs and perform more and more tests. However, this new incentive structure strikes us as being off the mark. If a doctor or a hospital is given a fixed amount of money to treat a certain patient (as is the case with many government sponsored health programs and health maintenance organizations) then doctors and hospitals wishing to maximize profit will perform fewer tests than necessary and generally undertreat the patient.

What is required is an incentive structure that aligns the natural interest of hospitals and doctors to make money with the desire of patients to be cured. Such a scheme might, for example, reward the practitioners whose ability to cure certain diseases is higher than the national average. A sliding scale could be implemented which would benefit both doctors and patients. All patients would pay the same amount for a treatment, but would then either be asked to pay more if the treatment was conspicuously successful or would have money refunded if the treatment was unsuccessful. Likewise, doctors and hospitals would be paid the same amount at the time of the treatment, but would be paid more if the treatment were successful and would be asked to return money if the treatment were unsuccessful.

Such an incentive strategy would discourage both too many and too few tests and encourage the use of appropriate technology. Hence, we would expect that under this incentive system, computerized methods which both decrease costs and increase effectiveness, such as the medical imaging system described in this chapter, would be widely used.

This incentive plan would require extensive computer support in order to keep track of doctors' records, to set reward schedules, and to oversee billing. Such a plan is unlikely to be implemented in the near future because it would require drastic changes in the medical infrastructure. Nevertheless, we feel that it would be one of the best ways of employing computation to improve health care.

Time line

The best test of a theory is to make predictions and see if they are correct. Although we do not have a theory about the future of medicine, but rather a loosely connected set of assumptions and heuristics, we can exercise them in the same way by making predictions.

Here are our predictions for the next two hundred years:

- 2000: Most patient records will be computerized, therefore increasing the opportunities for diagnostic computer systems to improve patient care.
- 2050: Completely automated robotic surgery will be commonplace.
- 2100: Human cloning and downloading (the process of copying a person's mind into a computer) will be possible. People who are gravely ill will simply have their memories and cognitive architectures piped into copies of their bodies.
- 2150: The convergence of a variety of computer technologies will conspire to make human lifespans effectively infinite.
- 2200: People will no longer have macro-physical instantiations; they

will be bits on a wire. Healing a person will have more in common with fixing a bug in a computer program than any other activity humans now engage in.

If these predictions turn out to be largely wrong, we suspect that this will be so because they are too conservative, not because they are exceedingly optimistic.

Singularities

The predictions given in the previous section are expected to hold only if certain computational advances, some of which are being actively investigated and others of which are only gleams in the eyes of futurists, do not intervene. Here we enumerate three of them and discuss how they might change our predictions. All three of them have the property such that if they are made available to large fractions of the population, their development will constitute historical singularities in the development of the human race.

- Artificial intelligence: The creation of an artificial intelligence, such as a thinking, conscious, free-willed electronic individual, will affect our basic notions about human evolution and progress, and will wreck any predictions made here. Indeed, humans in their present form may cease to exist and, therefore, any discussion of the role of computers in human medicine is moot.

- Downloading: Downloading is a procedure which places the human mind and all its attendant effects in a computer. We expect this to happen by the year 2200, but if it happens sooner, then the timeline given in the previous section will be accelerated.

- Nanotechnology: Drexler [6] has for over ten years imagined small machines that could be injected into the human body to cure disease. These small machines could be specifically engineered to target particular foreign cells and avoid negative side effects in a way that is unimaginable with currently available techniques, such as gene therapy and biotechnology.

Several other non-computer technologies, including cryogenics and human cloning, might also deeply influence the practice of medicine.

Conclusions

In the near-term, we expect that natural and easily foreseeable changes will take place in medicine. More patient records will be made available on-line and data mining algorithms [3] will cull through medical data and find new regularities. Therefore, diagnosis algorithms will become more sophisticated.

However, in the longer term (fifty years from now and more) we see changes that will drastically alter the practice of medicine and patient/doctor interaction. Artificial intelligence, downloading, and nanotechnology will cause qualitative changes in patient care.

Acknowledgments

This chapter describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research Contract N00014-91-J-4038.

Glossary

algorithm A precise set of instructions for accomplishing a task.

artificial intelligence The branch of computer science that deals with using computers to simulate human thinking.

cryogenics A means for freezing and maintaining a human body or a part of a human body (especially the head), with a view towards reviving it in the future.

CT scan Computer tomography scan.

cyberspace The electronic world, sometimes referred to as "virtual reality" when the user is deeply immersed in the images, sounds, etc.

human cloning A method for creating a copy of a human that is genetically identical to the original.

if-then rules A rule that has two parts, the first of which contains conditions that must be satisfied and the second of which contains assertions which will be true if the conditions are satisfied. Example: "If it is raining, then I will wear a raincoat."

genetic algorithm An optimization algorithm suggested by biological evolution and economic theory.

MRI Magnetic resonance imaging.

optimization algorithm An algorithm that locates the minima or maxima of a function.

simulated annealing An optimization algorithm inspired by a thermodynamic process.

Figure legends

Figure 1: A flow chart of a patient's progress through the medical system.

Figure 2: Alignment of MRI data (orange skull) with laser scan data (blue dots). This alignment is performed by the dynamic hill climbing algorithm.

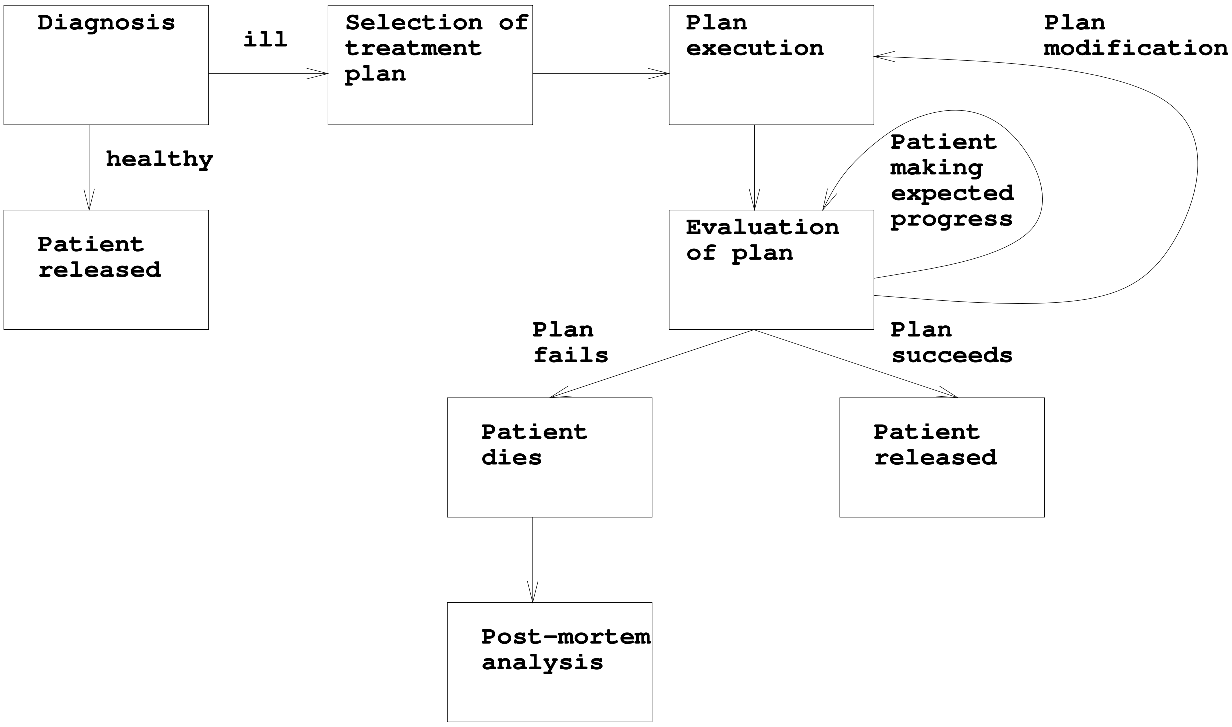

Figure 1:

Appendix: Inner loop code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DHC.LISP              ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; The following is an implementation of the dynamic hill climbing algorithm
;;; in LISP.  The DHC procedure expects four arguments:
;;;
;;; >> func: Function from float vectors to float values to be minimized.
;;; >> xini: A float vector specifying the initial starting point.
;;; >> vmin: A float vector specifying the minimum step size in each dimension.
;;;          This determines the precision of the search.
```

Figure 2:

```
;;;          Using a finer grained search usually helps but is more expensive.
;;; >> vmax: A float vector specifying the maximum step size in each dimension.
;;;          This determines how fast DHC can move in the search space.
;;;          When DHC gets stuck in a maximum it will try expanding its steps
;;;          to find a way out.  VMAX also determines when to stop trying.
;;;
;;; To achieve scale independence, DHC represents its steps in units of vmin.
;;; The step array has the form: (u0 u1 u2 ... ud -ud ... -u1 -u0); where
;;;   d is the number of dimensions,
;;;   ui (0 < i <= d) is the step vector in the i'th dimension
;;;   u0 is the incrementally adjusted gradient vector
;;;   -ui is the step in the opposite direction to ui
;;; A unit step corresponds to an actual step of vmin in the search space.
;;;
;;; When DHC gets stuck at an optimum, it returns the best point it has found.
;;; This might be a local optimum.  Thus a global search strategy should be
;;; used in conjunction with DHC for more reliable results.  DHC should be used
;;; for efficient local optimization.
;;;
;;; Order of function evaluations made by DHC in the sphere function is
;;;   O(DIMS*LOG(DIST)); where
;;;   DIMS is the dimensionality of space
;;;   DIST is the distance of initial point to global optimum in vmin units


                                                  ;;;
(defun dhc                                        ;;;
    (&key                                         ;;; >> PARAMETERS:
     (func #'sphere)                              ;;; target function
     (xini #(4.0 -5.0))                           ;;; initial starting point
     (vmin #(0.01 0.01))                          ;;; vector of minimum step size
     (vmax #(5.12 5.12))                          ;;; vector of maximum step size
     &aux                                         ;;; >> LOCAL VARIABLES:
     (dims (length xini))                         ;;; dimensionality of space
     (imax (+ (* 2 dims) 1))                      ;;; index of last step vector
     (fini (funcall func xini))                   ;;; value at initial point
     (smax (concatenate '(vector single-float)    ;;; the array of max step size
             (map 'list #'/ vmax vmin)            ;;; is ratio of vmax to vmin
```

```
          (list -1.0))))                     ;;; plus a -1 for extra vector
(do                                          ;;; >> STATE VARIABLES:
 ((stuck nil)                                ;;; flag set when stuck
  (xbst (copy-seq xini)) (fbst fini)         ;;; best point and its value
  (xcur (copy-seq xini)) (fcur fini)         ;;; current point and its value
  (step (init-step smax))                    ;;; array of int step vectors
  (xstep (init-step smax))                   ;;; step before last motion
  (gradv (v0 dims)) (gradi -1)               ;;; gradient vec and its index
  (last (v0 dims))                           ;;; last step taken
  (i -1))                                    ;;; index into step
 ((progn                                     ;;; >> TERMINATION TEST:
    (if stuck                                ;;; if stuck flag set
        (setf i (argmin step smax))          ;;;   choose the smallest step
        (setf i (argmax step)))              ;;;   else choose the largest
    (= i -1))                                ;;; if no more steps
  xbst)                                      ;;;   return the best point
                                             ;;; >> MAIN LOOP:
  (map-into xcur                             ;;; set xcur to the
          #'(lambda (x y z) (+ x (* y z)))   ;;;   best point plus the
          xbst (aref step i) vmin)           ;;;   step vector normalized
  (setf fcur (funcall func xcur))            ;;; set fcur to value at xcur
  (cond                                      ;;;
    ((< fcur fbst)                           ;;; >> IF NEW POINT BETTER
     (if stuck (setf stuck nil))             ;;; clear the stuck flag
     (replace xbst xcur) (setf fbst fcur)    ;;; update best point & val
     (dotimes (j (1+ imax))                  ;;; since we have moved
       (replace (aref xstep j) (aref step j)));;;   record step in xstep
     (map-into (aref step (- imax i))        ;;; set opp step to -curr/2
             #'-half (aref step i))          ;;;   not to retry same point
     (when (equalp (aref step i) last)       ;;; if last step repeated
       (map-into (aref step i)               ;;;   set the current step
               #'double (aref step i))       ;;;   to double the size
       (replace (aref xstep i) (aref step i)));;;   update xstep as well
     (replace last (aref step i))            ;;; record the last step
      (cond                                  ;;; Update the grad vector:
       ((= gradi -1)                         ;;;   if gradv is empty
        (replace gradv (aref step i))        ;;;     set gradv to curr step
        (setf gradi (min i (- imax i))))     ;;;     record the index
       ((= gradi (min i (- imax i)))         ;;;   else if curr vec is ||
```

```
          (map-into gradv                            ;;;     add the curr vec
                    #'+ gradv (aref step i)))        ;;;       to the gradv
         (t                                          ;;;   else update the extra vec
          (map-into (aref step dims)                 ;;;     set extra step to
                    #'+ (aref step i) gradv)         ;;;       gradv + curr vec
          (setf (aref smax dims)                     ;;;     set smax to the max of
            (max (aref smax (min i (- imax i)))      ;;;       curr vec smax
                 (aref smax gradi)))                 ;;;       and gradv smax
          (map-into (aref step (1+ dims))            ;;;     set the opposite step
                    #'- (aref step dims))            ;;;       to minus extra step
          (setf gradi -1))))                         ;;;     erase the gradv
      (stuck                                         ;;; >> ELSE IF ALREADY STUCK
       (map-into (aref step i)                       ;;; double the current step
                 #'double (aref step i)))            ;;;
      ((>= (size (aref step i)) 1.0)                 ;;; >> ELSE IF STEP >= VMIN
       (map-into (aref step i)                       ;;; half the current step
                 #'half (aref step i)))              ;;;
      (t                                             ;;; >> ELSE IF STEP TOO SMALL
       (setf stuck t)                                ;;; set the stuck flag
       (dotimes (j (1+ imax))                        ;;; reset the step vectors to
         (map-into (aref step j)                     ;;;   double the last recorded
                   #'double (aref xstep j)))))))     ;;;   step vectors
                                                     ;;;
                                                     ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Following is a list of definitions of the procedures called by DHC.
;;; >> SQ        : returns the square of a number
;;; >> VO        : takes # of dims and returns a zero vector
;;; >> SIZE      : calculates the size of a vector
;;; >> DOUBLE    : multiplies a number by two
;;; >> HALF      : divides a number by two
;;; >> -HALF     : minus the half of a number
;;; >> INIT-STEP : initializes step to unit vectors
;;; >> ARGMIN    : returns the minimum sized step within bounds in a step array
;;; >> ARGMAX    : returns the maximum sized step in a step array
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```lisp
(defun sq (x) (* x x))

(defun v0 (d) (make-array d :element-type 'single-float :initial-element 0.0))

(defun size (vec) (sqrt (apply #'+ (map 'list #'sq vec))))

(defun double (x) (* x 2))

(defun half (x) (/ x 2))

(defun -half (x) (- (half x)))

(defun init-step (smax)
  (let* ((dims (1- (length smax)))
         (imax (1+ (* 2 dims)))
         (step (make-array (* 2 (1+ dims)))))
    (dotimes (i (length step) step)
      (setf (aref step i) (v0 dims))
      (cond ((< i dims) (setf (aref (aref step i) i) (aref smax i)))
            ((<= i (1+ dims)))
            (t (setf (aref (aref step i) (- imax i))
                     (- (aref smax (- imax i)))))))))

(defun argmin (step smax)
  (let ((nstep (length step))
        (imin -1) (min MOST-POSITIVE-SINGLE-FLOAT))
    (dotimes (i nstep imin)
      (let ((si (size (aref step i))))
        (if (and (<= si (aref smax (min i (- nstep i 1)))) (< si min))
            (setf min si imin i))))))

(defun argmax (step)
  (let ((nstep (length step))
        (imax -1) (max 0.0))
    (dotimes (i nstep imax)
      (let ((si (size (aref step i))))
        (if (> si max)
          (setf max si imax i))))))
```

References

[1] J.S. Aikins, J.C. Kunz, E.H. Shortliffe, and R.J. Fallat. *Readings in Medical Artificial Intelligence: The First Decade*, chapter PUFF: An Expert System for Interpretation of Pulmonary Data. Addison-Wesley Publishing Company, Reading, MA, 1984.

[2] G. O. Barnett, J.J. Cimino, J.A. Hupp, and E.P. Hoffer. DXplain: An evolving diagnostic decision-support system. *Journal of the American Medical Association*, 258(1):67–74, 1987.

[3] I. Bhandari. Attribute focusing: Open-world data exploration aplied to software production process control. IBM RC 18320, International Business Machines, 1992.

[4] L. Davis, editor. *Genetic Algorithms and Simulated Annealing.* Morgan Kaufman, Los Altos, CA, 1987.

[5] M. de la Maza and D. Yuret. Dynamic hill climbing. *AI Expert*, 1994.

[6] E. K. Drexler, C. Peterson, and G. Pergamit. *Unbounding the Future: The Nanotechnology Revolution.* Morrow, New York, 1991.

[7] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[8] J.H. Grossman. Plugged-in medicine. *Technology Review*, 97(1):22–29, January 1994.

[9] J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[10] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[11] C.A Kulikowski and S.M. Weiss. *Artificial Intelligence in Medicine*, chapter Representation of Expert Knowledge for Consultation: The CASNET and EXPERT Projects. Westview Press, Boulder, CO, 1982.

[12] W.B. Schwartz. Medicine and the computer: The promise and problems of change. *New England Journal of Medicine*, 55:459–472, 1970.

[13] E. Shortliffe. Computer programs to support clinical decision making. *Journal of the American Medical Association*, 258(1):61–66, 1987.

[14] D. Yuret and M. de la Maza. Dynamic hill climbing: Overcoming the limitations of optimization techniques. In *Second Turkish Symposium on Artificial Intelligence and Artificial Neural Networks*, pages 208–212, 1993.